

# 武汉大学国际软件学院

## 实验报告

课程名称 Database System

专业年级 2013, Software Engineering

姓 名 王孜晗

学 号 2013302580150

协 作 者 \_\_\_\_\_

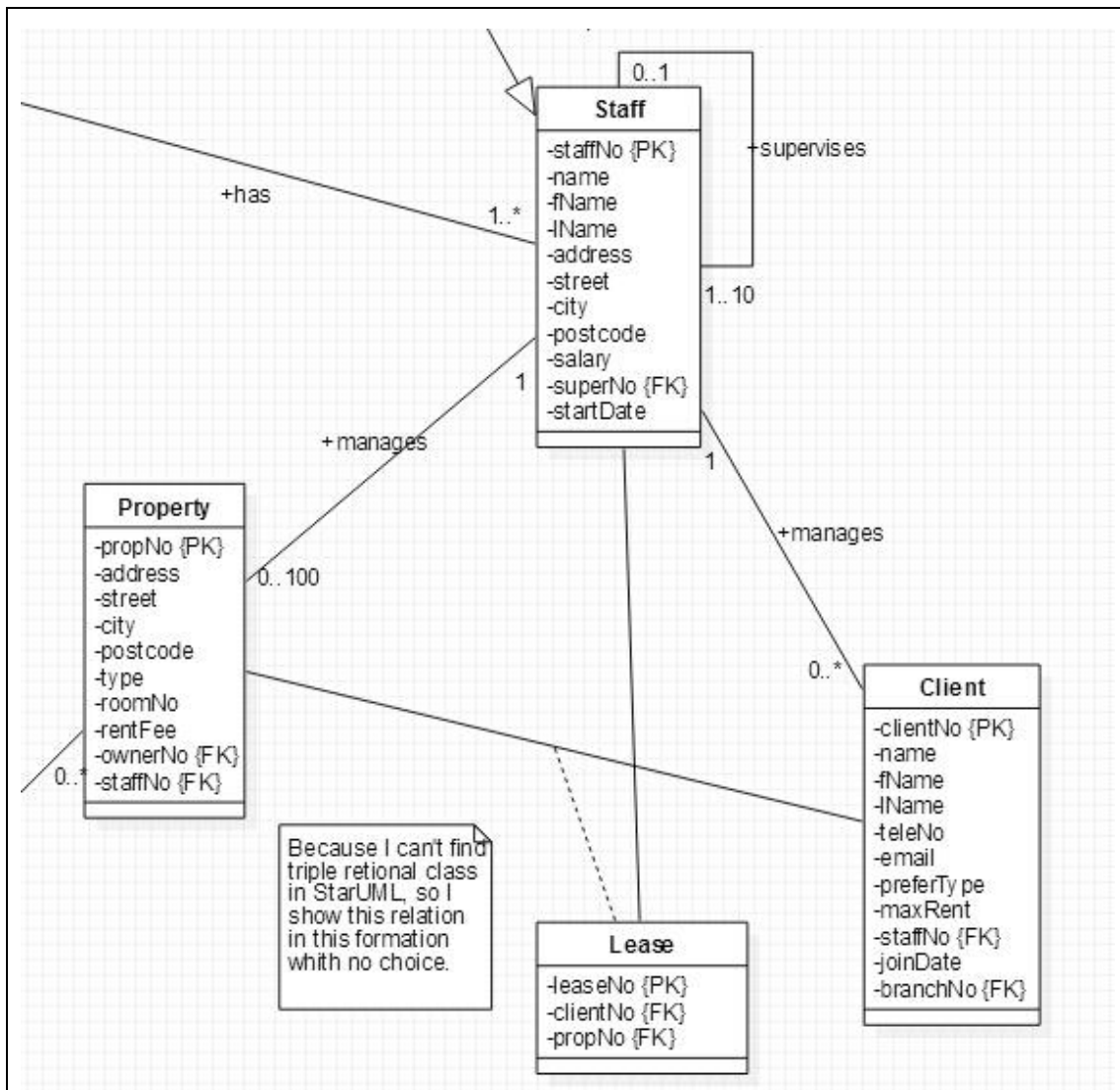
实验学期 2014-2015 学年 2 学期

课堂时数 \_\_\_\_\_ 课外时数 \_\_\_\_\_

填写时间 2015 年 06 月 17 日

实验概述
<p><b>【实验项目名称】：</b> Dream House Website SQL Program</p> <p><b>【实验目的】：</b></p> <ol style="list-style-type: none"> <li>1. Understand the overall development process of the JAVA Web project, including the database structure, the JDBC connection, the development code structure, and the physical deployment of the project;</li> <li>2. Learn to install jdk, tomcat, eclipse, postgresql and jdbc driver.</li> <li>3. Refer to the Appendix A or B to choose a case for development and analyze the user' s requirements specification for the case.</li> <li>4. Learn to draw the ER diagram for the case, build its logical data model, and normalize it.</li> <li>5. Learn to implement all transaction requirements in the case.</li> </ol> <p><b>【实验环境】（使用的软件）：</b></p> <p>PostgreSQL</p> <p>Navicat for PostgreSQL</p> <p>Java 8</p> <p>Tomcat 8</p> <p>IntelliJ IDEA 14.1.1</p> <p><b>【参考资料】：</b></p> <p>Database System: A Practical Approach to Design, Implementation and Management (Fifth Edition)</p> <p>《Java Web 开发实战经典》</p>
实验内容
<p><b>【实验方案设计】：</b></p> <p>I. Analyze the Dream House case, draw and normalize Enhanced E-R Mode</p> <p>After Analyzing the <i>Data Requirements</i> specified in APPENDIX A in our textbook, we find that:</p> <ol style="list-style-type: none"> <li>1. Manager of a branch is optionally generated from relation Staff and it contains one more fields: <i>bonus</i></li> <li>2. To normalize, when it comes to table <i>Branch</i>, we include field <i>managerNo</i> instead of <i>managerName</i>. When we want to query the manager's name of a specific branch, we can query from the join of table <i>Branch</i> and <i>Staff</i>.</li> </ol>

3. To normalize, when it comes to table *Staff*, we use field *superNo*, which is a foreign key pointing to itself, instead of *superName*. When we want to query the supervisor's name of a specific staff, we can query from table *Staff* using the foreign key *surperNo*.
4. To normalize, when it comes to table *Property*, we use field *ownerNo*, which is a foreign key pointing to a record in table *Owner*, instead of including all details of the relevant *owner*. When we want to query the owner's detail of a specific property, we can query from table *Owner* using the foreign key *ownerNo*.
5. To normalize, when it comes to table *Client*, we use field *branchNo*, which is a foreign key pointing to a record in table *Branch*, instead of including all details of the relevant *branch*. When we want to query the branch detail to which a specific property registered in, we can query from table *Branch* using the foreign key *branchNo*. What's more, we also use field *staffNo* instead of *staffName*. When we want to query the staff's name who manages a specific client, we can query from table *Staff* using the foreign key *staffNo*.
6. When it comes to viewing, we make viewing a relational class, say *Association Class*, to represent the table *Viewing*.  
Because I can't find triple retional class in StarUML, so I show this relation in this formation whith no choice.



7. To normalize, when it comes to table *Lease*(association class), we use field *clientNo*, which is a foreign key pointing to a record in table *Client*, instead of including all details of the relevant *client*. When we want to query the client detail for whom a specific property registered, we can query from table *Client* using the foreign key *clientNo*. In the same way, we also use *propNo* instead of including all detail of the relevant *property*.

Moreover, if we want to fetch the *amount of deposit*, we can use the foreign key to query the *rentFee* of the relevant record in *Property* and doubles it.

If we want to fetch the *duration* of the specific lease, we can use the *Date* value of the field *endDate* to subtract that of field *startDate*.

8. The similar thing of Step7 is also done to table *Ad* and *Viewing*.
- II. Install jdk, tomcat, IntelliJ, JDBC driver, struts2 and configure them correctly.
- Here I use IntelliJ IDEA instead of Eclipse to finish this experiment, as it's much easier to use. It can download jars and libraries that are needed for struts2 from the Internet automatically.
- For PostgreSQL JDBC driver, I download this:

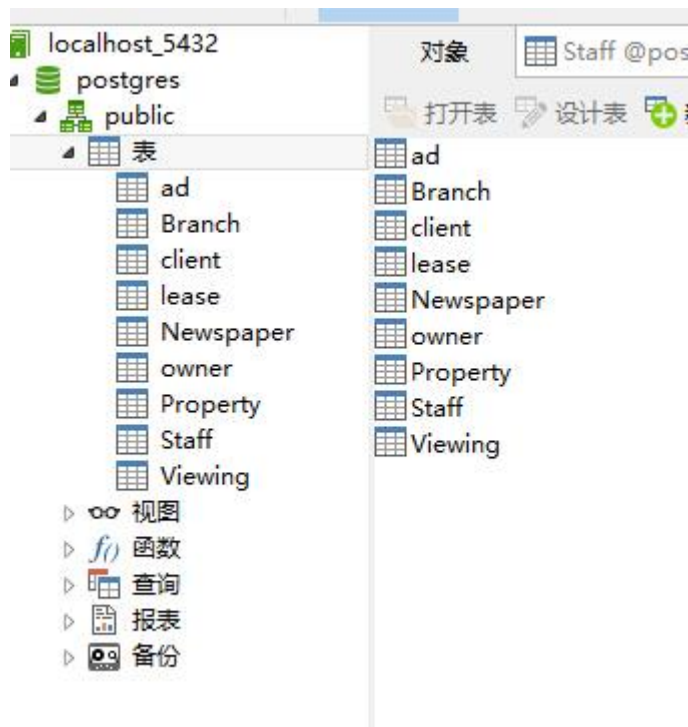
postgresql-9.4-1201.jdbc41.jar

And then I struggled to configure them correctly.

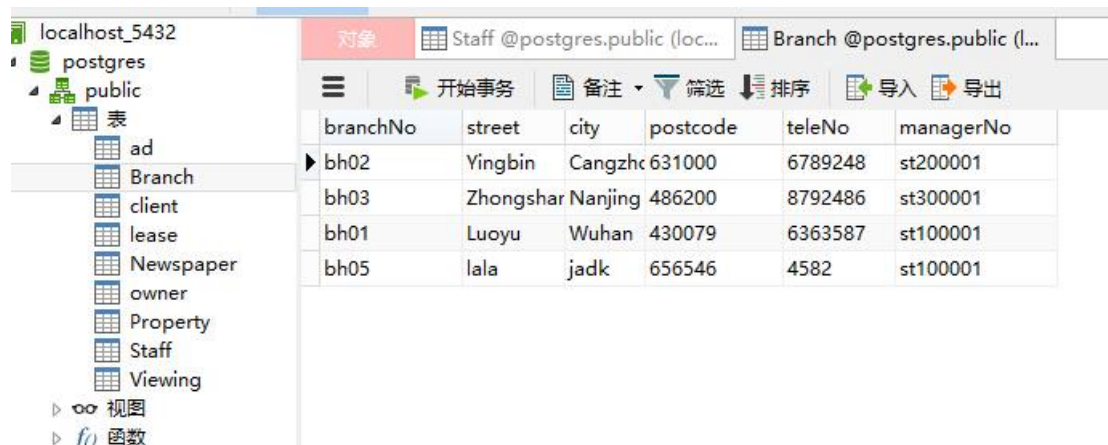
### III. Establish the database, create table and insert some demo data.

1. Before handle Dream House data through web, I first create tables according to the Enhanced Entity-Relationship Model drawn in the beginning.

To make this work easily to be done, I use Navicat For PostgreSQL instead of using pgAdmin III.



2. Then insert some demo data in the tables created just now.



### IV. Use structs2 to complete the insertion, deletion, updating and query transactions according to MVC model.

Here, I use the insertion of new Branch records to show the mechanism of realizing the database operations.

1. Login – Entity, loginAction, jsp and struct.xml

To login, first create a Admin class to receive the information <s:form>

transfers and then verify the user id and password in loginAction.java. But to make this work, we must configure action mapping in the struct.xml. In struts2, Java inflection mechanism is widely used.



```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>
    <package name="dreamhouse.action" extends="struts-default">
        <action name="LoginAciton" class="dream.action.LoginAction">
            <result name="success">
                /branch.jsp
            </result>
            <result name="input">
                /index.jsp
            </result>
        </action>
        <action name="LogoutAction" class="dream.action.LogoutAction">
            <result name="success">
                /index.jsp
            </result>
            <result name="input">
                /fail.html
            </result>
        </action>
    </package>
</struts>
```

```
inAction >
branch_query.jsp x struts.xml x LoginAction.java x client_property.jsp x

package dream.action;

/**
 * Created by lenovo on 2015/6/15.
 */
import ...

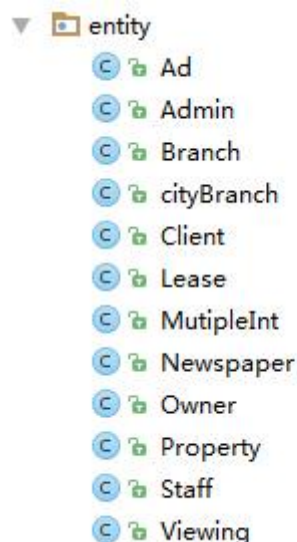
public class LoginAction implements Action{
    private Admin admin;

    @Override
    public String execute() throws Exception {
        if (("admin").equals(admin.getUser_id()) && ("admin").equals(admin.getPrd()))
            return SUCCESS;
        else
            return INPUT;
    }

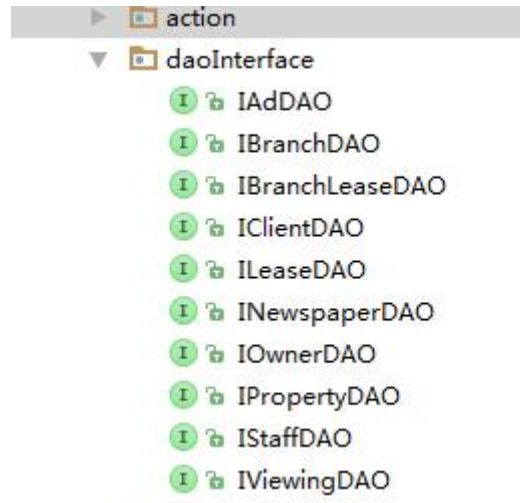
    public Admin getAdmin() { return admin; }

    public void setAdmin(Admin admin) { this.admin = admin; }
}
```

2. Create entity class for all the tables with private attributes for later using. Remember to write the getters and setters.



3. Create DAO Interface for each entity DAO, which includes all the methods that are needed for one table. This step is done for later using to create concrete DAOs.



4. Create DatabaseConnection class to create connection instances which connects to PostgreSQL. This step uses the JDBC Driver for PostgreSQL we downloaded in the previous step.

```
DatabaseConnection.java x
/* Created by Ienovo on 2015/6/15. */
public class DatabaseConnection {
    private static final String DBDRIVER = "org.postgresql.Driver";
    private static final String DBURL = "jdbc:postgresql://localhost:5432/postgres";
    private static final String DBUSER = "postgres";
    private static final String DBPASSWORD = "123698745qqqbvay";
    private Connection coon = null;
    public DatabaseConnection() {
        try {
            Class.forName(DBDRIVER);
            this.coon = DriverManager.getConnection(DBURL, DBUSER, DBPASSWORD);
            System.out.println("Opened database successfully");
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println(e.getClass().getName() + ": " + e.getMessage());
        }
    }

    public Connection getConnection() { return this.coon; }

    public void close() throws Exception {
        if(this.coon != null){
            try {
                this.coon.close();
            } catch (Exception e) {
                throw e;
            }
        }
    }
}
```

5. Create DAOs for the relations we want to manipulate. In this DAOs include the concrete SQL operations. Remember to write the constructor functions with database connection.



▼ db

- AdDAO
- BranchDAO
- BranchLeaseDAO
- ClientDAO
- DatabaseConnection
- dbcp.properties
- LeaseDAO
- NewspaperDAO
- OwnerDAO
- PropertyDAO
- StaffDAO
- ViewingDAO

```
DatabaseConnection.java x BranchDAO.java x
/* Created by lenovo on 2015/6/15.
*/
public class BranchDAO implements IBranchDAO {
    private Connection conn = null;
    private PreparedStatement pstmt = null;

    public BranchDAO(Connection conn) { this.conn = conn; }

    @Override
    public boolean insertBranch(Branch branch) throws Exception {
        boolean flag = false;
        try {
            System.out.println("start Branch");
            String sql = "SELECT * FROM \"Branch\" WHERE \"branchNo\"=?";
            this.pstmt = this.conn.prepareStatement(sql);
            this.pstmt.setString(1, branch.getBranchNo());
            ResultSet rs = this.pstmt.executeQuery();
            if (!rs.next()) {
                System.out.println("find Branch");
                sql = "INSERT INTO \"Branch\" VALUES (?, ?, ?, ?, ?, ?)";
                this.pstmt = this.conn.prepareStatement(sql);
                this.pstmt.setString(1, branch.getBranchNo());
                this.pstmt.setString(2, branch.getStreet());
                this.pstmt.setString(3, branch.getCity());
                this.pstmt.setString(4, branch.getPostcode());
                this.pstmt.setString(5, branch.getTeleNo());
                this.pstmt.setString(6, branch.getManagerNo());
                this.pstmt.executeUpdate();
                flag = true;
            }
        } catch (Exception e) {
```

6. Write DAOProxy classes to initiate relevant DAOs with DatabaseConnection instances and do indeed queries.

proxy

AdDAOProxy

BranchDAOProxy

BranchLeaseDAOProxy

ClientDAOProxy

LeaseDAOProxy

NewspaperDAOProxy

OwnerDAOProxy

PropertyDAOProxy

StaffDAOProxy

ViewingDAOProxy

BranchDAOProxy.java

```

/**
 * Created by lenovo on 2015/6/15.
 */
public class BranchDAOProxy implements IBranchDAO {
    private DatabaseConnection dbc = null;
    private IBranchDAO dao = null;

    public BranchDAOProxy() {

        this.dbc = new DatabaseConnection();
        this.dao = new BranchDAO(this.dbc.getConnection());
    }

    public boolean insertBranch(Branch branch) throws Exception {
        boolean flag = false;
        try{
            flag = this.dao.insertBranch(branch);
        }catch (Exception e) {
            throw e;
        }
        return flag;
    }

    @Override
    public LinkedList<Branch> getAllBranch() throws Exception {
        return this.dao.getAllBranch();
    }

    @Override
    public boolean updateBranch(Branch branch) throws Exception {

```

7. Create a DAOFactory class to gather all the DAOProxy classes and therefore we can use them more easily in the later process.

BranchDAOProxy.java x
DAOFactory.java x

```

package dream.factory;

import ...

/**
 * Created by lenovo on 2015/6/15.
 */
public class DAOFactory {
    public static IBranchDAO getIBranchDAOInstance() { return new BranchDAOProxy(); }
    public static IStaffDAO getIStaffDAOInstance() { return new StaffDAOProxy(); }
    public static IAdDAO getIAdDAOInstance() { return new AdDAOProxy(); }
    public static IClientDAO getIClientDAOInstance() { return new ClientDAOProxy(); }
    public static ILeaseDAO getILeaseDAOInstance() { return new LeaseDAOProxy(); }
    public static INewspaperDAO getINewspaperDAOInstance() { return new NewspaperDAOProxy(); }
    public static IOwnerDAO getIOwnerDAOInstance() { return new OwnerDAOProxy(); }
    public static IPropertyDAO getIPropertyDAOInstance() { return new PropertyDAOProxy(); }
    public static IViewingDAO getIViewingDAOInstance() { return new ViewingDAOProxy(); }
    public static IBranchLeaseDAO getIBranchLeaseDAOInstance() {
        return new BranchLeaseDAOProxy();
    }
}

```

8. Write the Relevant jsp file of the specific query, for example, branch\_insert.jsp for Branch record insertion.

BranchDAOProxy.java x DAOFactory.java x branch\_insert.jsp x

jsp:root s:form

```

<div class="ct-time">
  <ul class="separator-class">

  </ul>
</div>
<div class="courses">
  <div class="separator">
    <s:form action="insertBranchAction" method="post">
      <ul class="separator-class">

        <li><s:textfield name="branch.branchNo" tooltip="Branch NO"
          class="tf" placeholder="branch No"/></li>
        <li><s:textfield name="branch.street" tooltip="street"
          class="tf" placeholder="street" /></li>
        <li><s:textfield name="branch.city" tooltip="city"
          class="tf" placeholder="city" /></li>
        <li><s:textfield name="branch.postcode" tooltip="postcode"
          class="tf" placeholder="postcode" /></li>
        <li><s:textfield name="branch.teleNo" tooltip="telephone"
          class="tf" placeholder="teleNo" /></li>
        <li><s:textfield name="branch.managerNo" tooltip="managerNo"
          class="tf" placeholder="managerNo" /></li>
        <li><s:submit id="submit" class="tf" value="insert" /></li>
      </ul>
    </s:form>
  </div>
</div>

```

9. Write the relevant Action to receive the input data and handle this requirement.

```

import ...

/**
 * Created by lenovo on 2015/6/16.
 */

public class InsertBranchAction implements Action {
    private Branch branch;

    public Branch getBranch() { return branch; }

    public void setBranch(Branch branch) { this.branch = branch; }

    @Override
    public String execute() throws Exception {
        System.out.println("executing!");
        DAOFactory factory = new DAOFactory();
        IBranchDAO branchDAO = factory.getIBranchDAOInstance();
        boolean flag = branchDAO.insertBranch(branch);
        if(flag){
            return SUCCESS;
        }
        return INPUT;
    }
}

```

10. Finally, before using, we must configure the struct.xml.

```

<action name="insertBranchAction" class="dream.action.InsertBranchAction">
    <result name="success">
        /branch.jsp
    </result>
    <result name="input">
        /fail.html
    </result>
</action>

```

V. Use form and jsp to complete some other query transactions.

As I encounter some problems when using servlet, to simplify my work, I just use jsp to handle the queries acquired besides insertions, deletions and updating. What's more, querying for all the details of a specific table is also accomplished jsp.

As the mechanisms of these queries are similar to each other, so here I just take transaction *Find Properties Matching A Client* as an example. For more details of other queries, the source code of my project is available.

1. Create a form in the initial query jsp (branch\_query.jsp) which can transfer the value input from this initial page to the destination jsp file.



```

<h3>Find Properties Matching A Client</h3>
<ul class="separator-class">
  <form action="client_property.jsp" method="post">
    <li><input type="text" name="client" placeholder="clientNo" /> </li>
    <li><input type="submit" value="Query"></li>
  </form>
</ul>

```

- To finish this query function, we first go to modify the IPropertyDAO.java to add a new function for this query.

```

/*
public interface IPropertyDAO {
    public boolean insert(Property property) throws Exception;
    public LinkedList<Property> getAll() throws Exception;
    public boolean update(Property property) throws Exception;
    public boolean delete(Property property) throws Exception;
    public boolean find(Property property) throws Exception;
    public LinkedList<MutipleInt> rentBranch() throws Exception;
    public LinkedList<Property> propertyWant(int room, double rent) throws Exception;
    public LinkedList<MutipleInt> staffProperty(String branchNo) throws Exception;
    public LinkedList<Property> clientProperty(String clientNo) throws Exception;
}

```

- Then we go the PropertyDAO.java to implement the function we just add in the interface of PropertyDAO.

```

PropertyDAO.java
@Override
public LinkedList<Property> clientProperty(String clientNo) throws Exception {
    LinkedList<Property> result = new LinkedList<>();
    try {
        String sql= "SELECT * FROM \"Property\" p WHERE EXISTS(SELECT * FROM \"client\" c WHERE \"ClientNo\"=? +
            \" AND p.\"type\" = c.\"type\" AND p.\"rentFee\" <= c.\"maxRent\")";

        this.pstmt = this.conn.prepareStatement(sql);
        this.pstmt.setString(1, clientNo);
        ResultSet rs = pstmt.executeQuery();
        while (rs.next()) {
            Property row = new Property();
            row.setPropNo(rs.getString(1));
            row.setStaffNo(rs.getString(2));
            row.setCity(rs.getString(3));
            row.setPostcode(rs.getString(4));
            row.setType(rs.getString(5));
            row.setRoomHave(rs.getInt(6));
            row.setRentFee(rs.getDouble(7));
            row.setOwnerNo(rs.getString(8));
            row.setStaffNo(rs.getString(9));
            result.add(row);
        }
    } catch (SQLException e) {
        throw e;
    } finally {
        if (this.pstmt != null) {
            try {
                this.pstmt.close();
            } catch (Exception e) {
                throw e;
            }
        }
    }
}

```

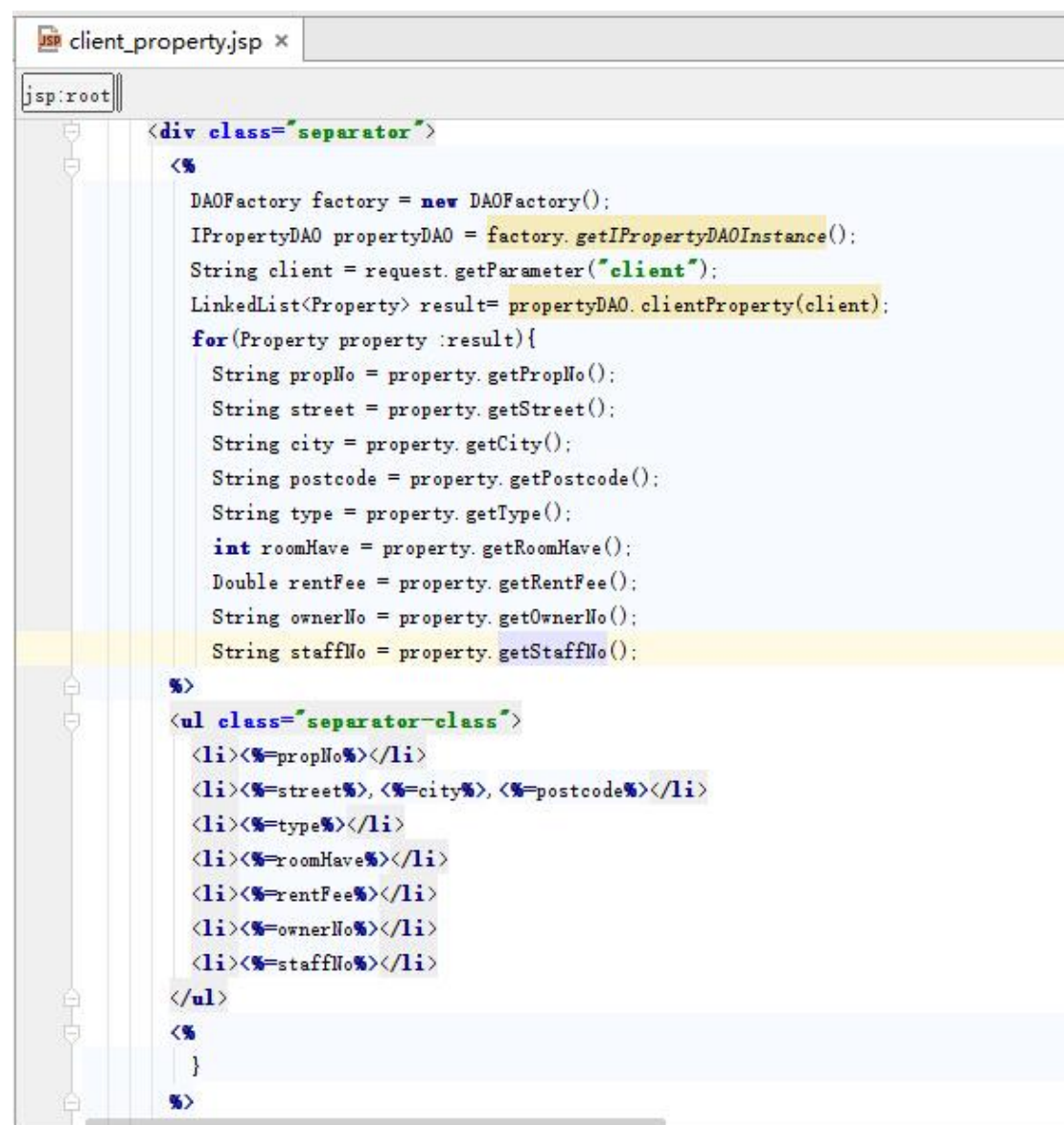
What's important is the String sql, which is responsible for the SQL query and get the correct result set.

4. Then go to PropertyDAOProxy.java to implement the function we just add in the interface.

```
@Override
public LinkedList<Property> clientProperty(String clientNo) throws Exception {
    return this.dao.clientProperty(clientNo);
}

}
```

5. That's all for the database relevant modification. Finally we create the jsp file response to the query acquirement.



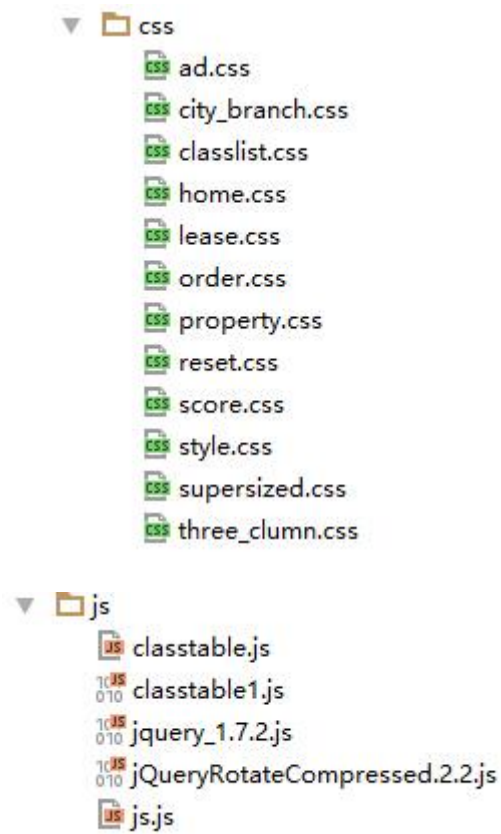
```
client_property.jsp
jsp:root
<div class="separator">
<%
    DAOFactory factory = new DAOFactory();
    IPropertyDAO propertyDAO = factory.getPropertyDAOInstance();
    String client = request.getParameter("client");
    LinkedList<Property> result = propertyDAO.clientProperty(client);
    for(Property property : result){
        String propNo = property.getPropNo();
        String street = property.getStreet();
        String city = property.getCity();
        String postcode = property.getPostcode();
        String type = property.getType();
        int roomHave = property.getRoomHave();
        Double rentFee = property.getRentFee();
        String ownerNo = property.getOwnerNo();
        String staffNo = property.getStaffNo();
    }
%>
<ul class="separator-class">
    <li><%=propNo%></li>
    <li><%=street%>, <%=city%>, <%=postcode%></li>
    <li><%=type%></li>
    <li><%=roomHave%></li>
    <li><%=rentFee%></li>
    <li><%=ownerNo%></li>
    <li><%=staffNo%></li>
</ul>
<%
}
%>
```

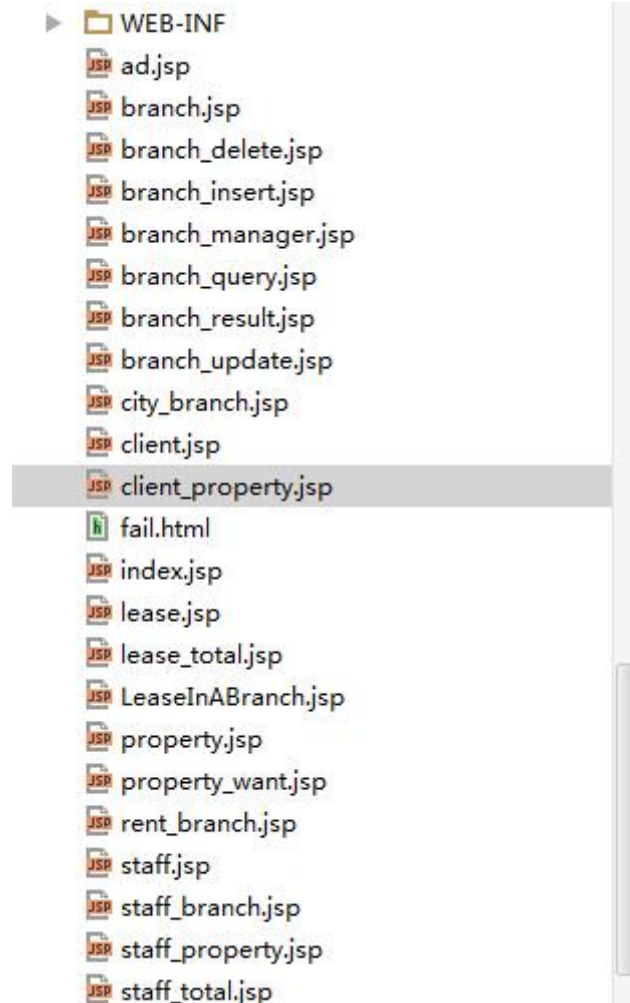
6. More technical details:

js and css files included to make the UI more attractive, although it really doesn't



seem attractive(23333).





### 【结论】（结果）：

During the experiment, I have encountered some bugs. I tried to fix the problems by asking friends for help or searching on the Internet for solutions. The problems and their solutions are as follows.

#### I. Fail to delete a row from a table because *the relation is not found*.

I searched on the Internet for solution but I can't fix it. Finally I asked my friend for help and she told me that PostgreSQL would turn the table name input into lower case and I have to use \"\" to surround the table name. It really works!

```
try {
    String sql= "SELECT * FROM \"Branch\"";
    this.pstmt = this.conn.prepareStatement(sql);
    ResultSet rs = pstmt.executeQuery();
```

#### II. Fail to insert a row into a table because of forgetting to type the brackets.

I tried thousands of times to insert a new record of branch through the web but it always failed and the server said that the error was in the following line,

```

System.out.println("find Branch");
sql = "INSERT INTO \Branch\" VALUES ?, ?, ?, ?, ?, ?";
this.pstmt = this.conn.prepareStatement(sql);
this.pstmt.setString(1, branch.getBranchNo());
this.pstmt.setString(2, branch.getStreet());
this.pstmt.setString(3, branch.getCity());
this.pstmt.setString(4, branch.getPostcode());
this.pstmt.setString(5, branch.getTeleNo());
this.pstmt.setString(6, branch.getManagerNo());
this.pstmt.executeUpdate();
flag = true;

```

I wondered why for a rather long time and can't figure it. Then I asked my friend for help again and she said that I had forgotten the parentheses after VALUES in the sql String. How stupid I was for making such a mistake!

```

if (!rs.next()) {
    System.out.println("find Branch");
    sql = "INSERT INTO \Branch\" VALUES ?, ?, ?, ?, ?, ?";
    this.pstmt = this.conn.prepareStatement(sql);
    this.pstmt.setString(1, branch.getBranchNo());
    this.pstmt.setString(2, branch.getStreet());
    this.pstmt.setString(3, branch.getCity());
    this.pstmt.setString(4, branch.getPostcode());
    this.pstmt.setString(5, branch.getTeleNo());
    this.pstmt.setString(6, branch.getManagerNo());
    this.pstmt.executeUpdate();
    flag = true;
}

```

So after add parentheses to the sql String, it can work!

```

if (!rs.next()) {
    System.out.println("find Branch");
    sql = "INSERT INTO \Branch\" VALUES (?, ?, ?, ?, ?, ?)";
    this.pstmt = this.conn.prepareStatement(sql);
    this.pstmt.setString(1, branch.getBranchNo());
    this.pstmt.setString(2, branch.getStreet());
    this.pstmt.setString(3, branch.getCity());
    this.pstmt.setString(4, branch.getPostcode());
    this.pstmt.setString(5, branch.getTeleNo());
    this.pstmt.setString(6, branch.getManagerNo());
    this.pstmt.executeUpdate();
    flag = true;
}

```

III. Fail to select a row from a table because *the field is not found*.

The similar error to the error I, where I didn't use \" \" to surround a field. Also we need to use \' \' to surround a value.

IV. Fail to connect to the database because the jdbc driver is no found.

When I first started the project, I found that I can't connect to the database. And the

error is print out as follows.

```
Output
[2015-06-15 09:24:55,148] Artifact DreamHouse:war exploded: Deploy took 3,004 milliseconds
15-Jun-2015 21:25:01.661 INFO [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Deploying web
15-Jun-2015 21:25:01.816 INFO [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of
java.lang.ClassNotFoundException: org.postgresql.Driver
    at org.apache.catalina.loader.WebappClassLoaderBase.loadClass(WebappClassLoaderBase.java:1305)
    at org.apache.catalina.loader.WebappClassLoaderBase.loadClass(WebappClassLoaderBase.java:1157)
    at java.lang.Class.forName0(Native Method)
    at java.lang.Class.forName(Class.java:264)
    at dream.db.DatabaseConnection.<init>(DatabaseConnection.java:17)
    at dream.proxy.BranchDAOProxy.<init>(BranchDAOProxy.java:19)
    at dream.factory.DAOFactory.getIBranchDAOInstance(DAOFactory.java:11)
    at org.apache.jsp.branch_jsp._jspService(branch_jsp.java:535)
```

A friend helped me to solve this problem. He told me to put the driver jar file in the lib directory of tomcat. And it works!

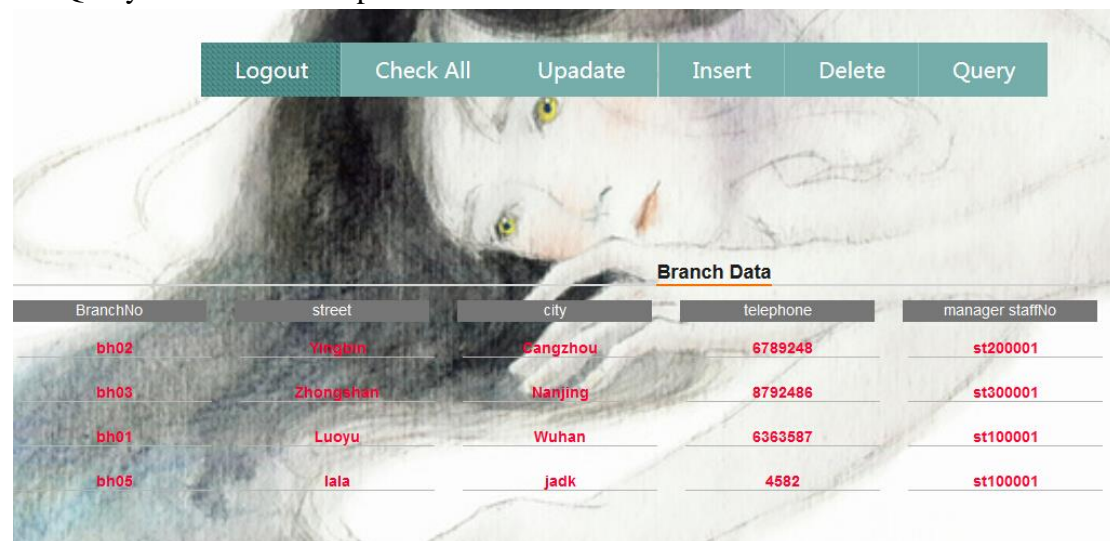
V. Fail to insert a row into a table because of typing the Action name wrongly.

I try hard to figure why the result is wrong and my Web teacher help me to solve this problem.

VI. The results of insertion, deletion, updating and Query the details of a specific table.

Here I take table *Branch* as an example.

1. Query the details of a specific table



The screenshot shows a web application interface. At the top, there is a navigation bar with six buttons: Logout, Check All, Update, Insert, Delete, and Query. Below the navigation bar is a table titled "Branch Data". The table has five columns: BranchNo, street, city, telephone, and manager staffNo. The table contains four rows of data.

BranchNo	street	city	telephone	manager staffNo
bh02	Yagbin	Gangzhou	6789248	st200001
bh03	Zhongshan	Nanjing	8792486	st300001
bh01	Luoyu	Wuhan	6363587	st100001
bh05	lala	jadk	4582	st100001

2. Insert a new Branch

First enter the data to insert.

After press the button *insert* we can jump to see the *Branch* data in the database now.

BranchNo	street	city	telephone	manager staffNo
bh02	Yagouin	Bangzhou	6789248	st200001
bh03	Zhongshan	Nanjing	8792486	st300001
bh01	Luoyu	Wuhan	6363587	st100001
bh05	lala	jadk	4582	st100001
bh07	lala	ajdfalk	486218	st100001

### 3. Update a Branch

First enter the data want to update.



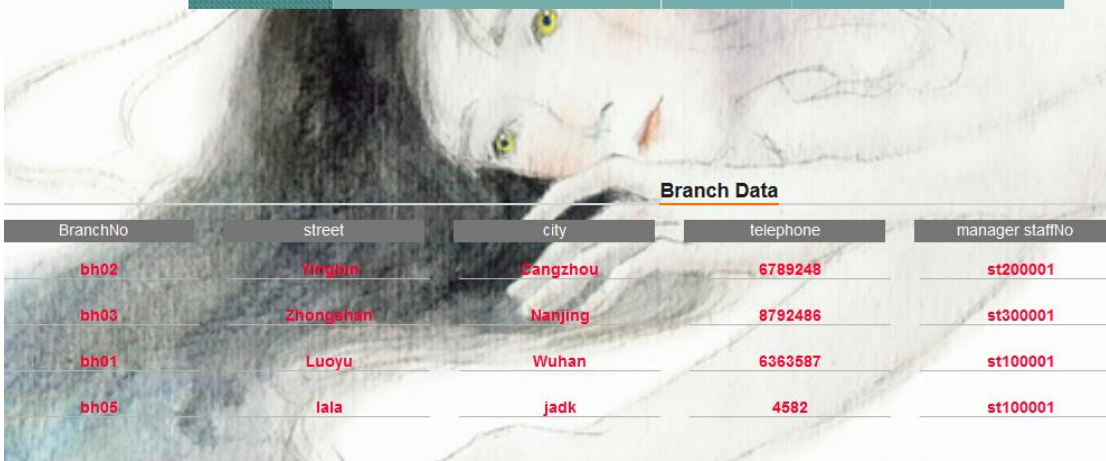
After press the button *insert* we can jump to see the *Branch* data in the database now.

Logout	Check All	Update	Insert	Delete	Query
<b>Branch Data</b>					
BranchNo	street	city	telephone	manager staffNo	
bh02	Yingbin	Hangzhou	6789248	st200001	
bh03	Zhongshan	Nanjing	8792486	st300001	
bh01	Luoyu	Wuhan	6363587	st100001	
bh05	lala	jadk	4582	st100001	
bh07	fafadf	fadg	4578	st100002	

#### 4. Delete a Branch

First enter the data want to delete.

After press the button *insert* we can jump to see the *Branch* data in the database now.

Logout	Check All	Update	Insert	Delete	Query
					
Branch Data					
BranchNo	street	city	telephone	manager staffNo	
bh02	Wagbin	Cangzhou	6789248	st200001	
bh03	Zhongshan	Nanjing	8792486	st300001	
bh01	Luoyu	Wuhan	6363587	st100001	
bh05	lala	jadk	4582	st100001	

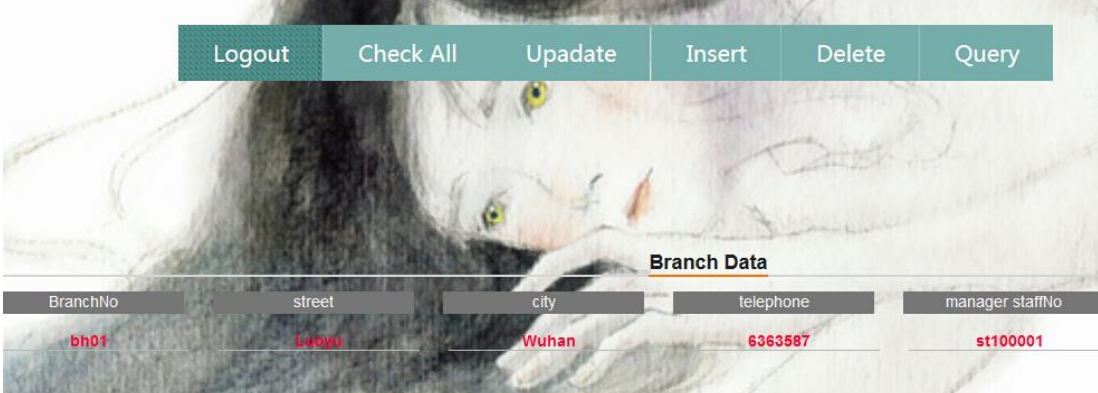
VII. Up to now, I will show the result of all my transactions realized.

### 1. Find Branch With Keyword

**Query Data**

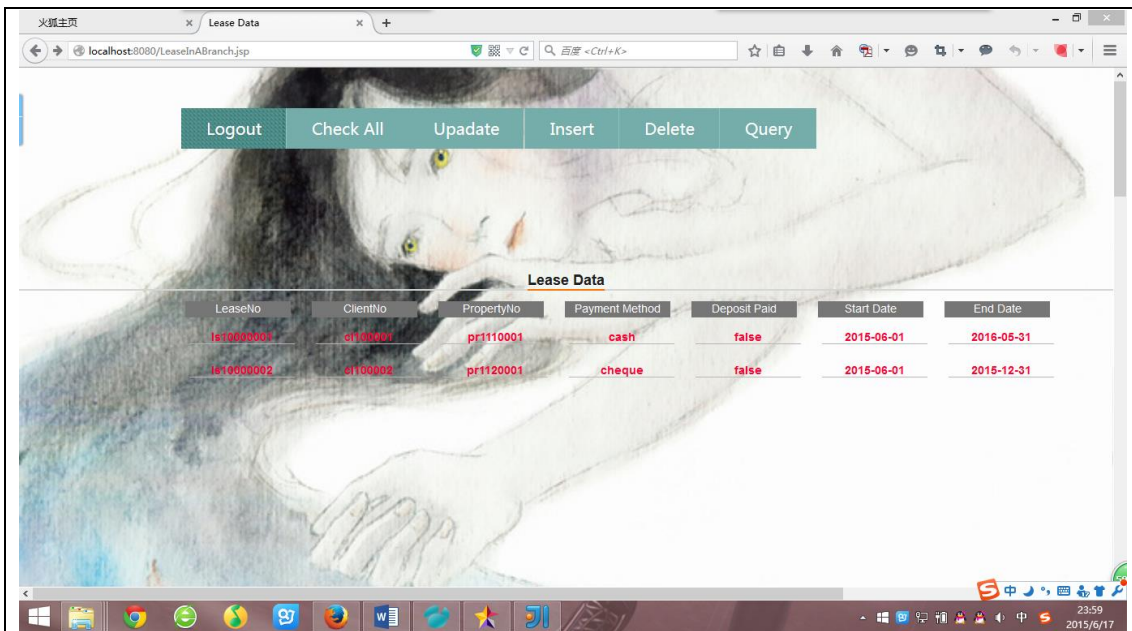
Find Branch With Keyword



Logout	Check All	Update	Insert	Delete	Query
					
Branch Data					
BranchNo	street	city	telephone	manager staffNo	
bh01	Luoyu	Wuhan	6363587	st100001	

### 2. Find Lease Details In A Branch

**Find Lease Details In A Branch**



### 3. Figure Out Total Number of Branch In Each City

Figure Out Total Number of Branch In Each City

Query

The screenshot shows a web browser window with the address bar displaying 'localhost:8080/LeaseInABranch.jsp'. The page has a navigation bar with buttons: Logout, Check All, Update, Insert, Delete, and Query. Below the navigation bar is a table titled 'Branch Data' with the following columns: City and Total Branch. The table contains four rows of data.

City	Total Branch
Wuhan	1
Cangzhou	1
Nanjing	1
jack	1

### 4. List Staffs In A Given Branch, Ordered By Name

List Staffs In A Given Branch

bh01

Query



Staff Data

StaffNo	Name	gender	DOB	Address	Salary	SuperManagerNo	Position
st100000	Amy Smith	F	1980-03-14	Luoja, Wuhan, 430079	15000.0	st100001	supervisor
st100001	Steven Smith	M	1972-05-13	Luoja, Wuhan, 430079	35000.0	null	manager
st100003	Tommy Stack	M	1983-12-14	Xionghu, Wuhan, 430079	8000.0	st100002	staff

## 5. Total Number Of Staff And Their Salaries Sum

Total Number Of Staff And Their Salaries Sum

Query

Query Result

Total Staffs	Total Salaries
3	194000

## 6. Manager Name Of Each Branch, Ordered By Address

Manager Name Of Each Branch

Query

Logout

Check All

Update

Insert

Delete

Query Result

Manager Name	BranchNo	City
Sue Clinton	bh02	Cangzhou
Steven Smith	bh05	jadk
Jack Jones	bh03	Nanjing
Steven Smith	bh01	Wuhan

#### 7. Total Max Rent Per Month For Each Branch

Total Max Rent Per Month For Each Branch

Query

Logout

Check All

Update

Insert

D

Query Result

BranchNo	Rent Fee Total
bh02	4800.0
bh03	5200.0
bh01	4600.0

8. Total Lease Endure Less for One Year In Each Branch

Total Lease Less for One Year In Each Branch

Query

Logout Check All Update Insert

**Query Result**

BranchNo	Total Lease
bh01	1
bh02	2
bh03	1

9. Property With Room No Less Than And Rent Fee No Higher Than

Property With Room No Less Than And Rent Fee No Higher Than

2

1500

Query



Property Data

localhost:8080/property\_want.jsp

Logout Check All Update Insert Delete Query

**Property Data**

PropertyNo	Address	Type	Rooms Number	Rent Per Month	OwnerNo	StaffNo
pr2110001	add,Changshou,531900	flat	2	1000.0	on210001	st200002
pr3120001	add,Nanjing,485200	flat	2	1400.0	on320001	st300003
pr1110001	null,Wuhan,430079	flat	2	1300.0	on110001	st100003

#### 10. Total Properties Assinged To Each Staff At A Given Branch

Total Properties Assinged To Each Staff At A Given Branch

bh01

Query

**Query Result**

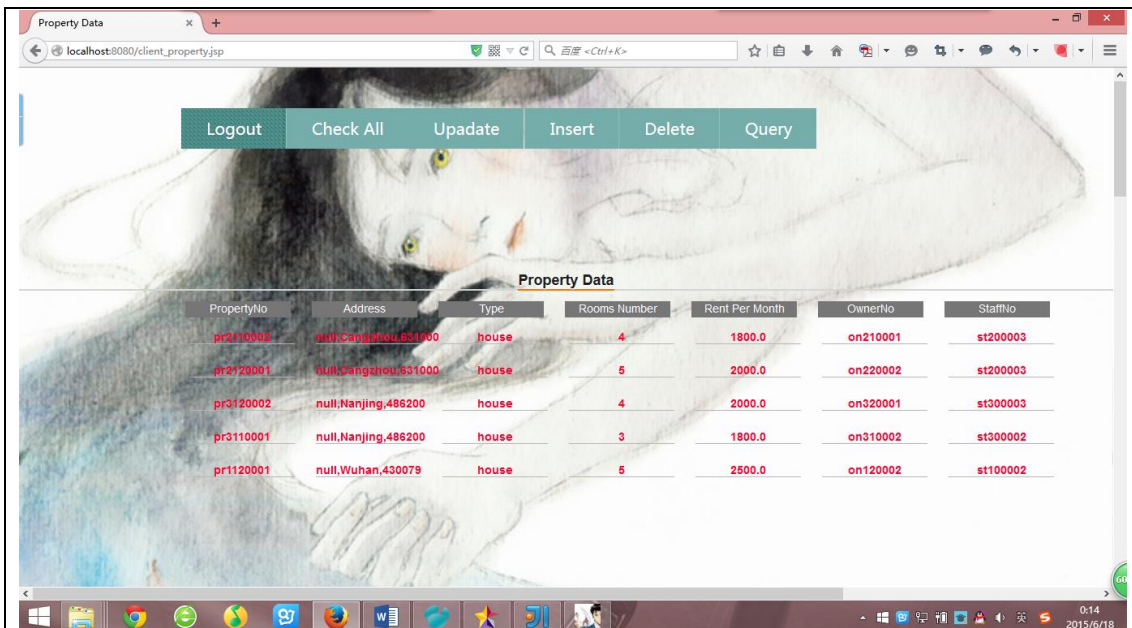
BranchNo	StaffNo	Properties Managing
bh01	st100002	1
bh01	st100003	2

#### 11. Find Properties Matching A Client

Find Properties Matching A Client

cl200002

Query



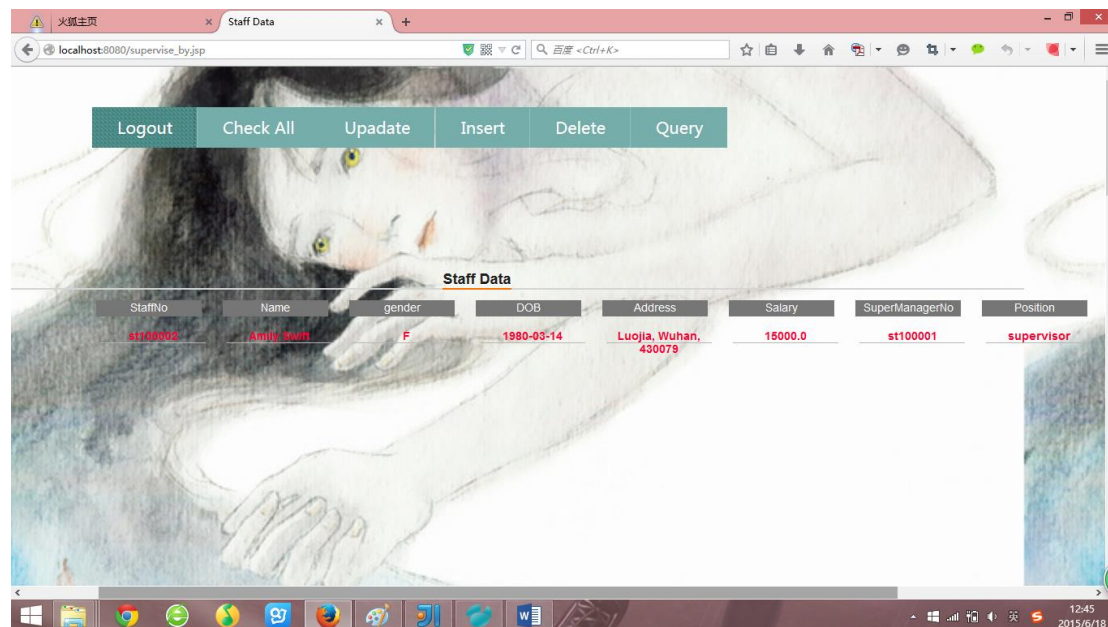
12. List details of staff supervised by a named Supervisor at the branch

List details of staff supervised by a named Supervisor at the branch

Amily

Swift

Query



13. List the details of properties that have not been rented out for more than three months

List the details of properties that have not been rented out for more than three months

Query



Property Data

localhost:8080/not\_rent\_for\_three.jsp

Logout Check All Update Insert Delete Query

Property Data

PropertyNo	Address	Type	Rooms Number	Rent Per Month	OwnerNo	StaffNo
pr2110002	null,Changshou,531900	house	4	1800.0	on210001	st200003
pr3120001	Null,Nanjing,486200	flat	2	1400.0	on320001	st300003
pr3120002	null,Nanjing,486200	house	4	2000.0	on320001	st300003
pr1110002	null,Wuhan,430079	flat	1	800.0	on110001	st100003

14. Produce a list of clients whose preferences match a particular property

Produce a list of clients whose preferences match a particular property

Query

Client Data

localhost:8080/match\_client.jsp

Logout Check All Update Insert Delete Query

Client Data

ClientNo	Name	Telephone	Email	Prefer Type	Max Rent can offer	StaffNo	Join Date	BranchNo
cl200002	Bouglby King	15078924863	dk@126.com	house	2500.0	st200002	2014-12-30	bh02
cl200001	Bobby Walker	13078928964	lls@163.com	flat	1200.0	st200003	2015-05-01	bh02
cl200003	Pheobe Baker	13513278501	ph@vip.qq.com	house	2700.0	st200003	2014-11-18	bh02
cl300003	Ultraman Cook	13078519647	ultra@qq.com	house	3000.0	st300002	2015-04-16	bh03
cl300001	Jerry Green	15804863782	je@hotmail.com	flat	1500.0	st300003	2014-12-18	bh03
cl300002	Mason Collins	13948524466	mson@gmail.com	house	3500.0	st300003	2015-03-19	bh03
cl100003	John Young	15015987015	125@qq.com	house	3000.0	st100002	2014-11-23	bh01
cl100002	Hank Hall	13578968414	abc@gmail.com	flat	1500.0	st100003	2014-12-03	bh01
cl100001	Bill Levis	15479234861	123@hotmail.com	flat	2000.0	st100003	2015-03-15	bh01

15. Display the details of a lease between a named client and a given property

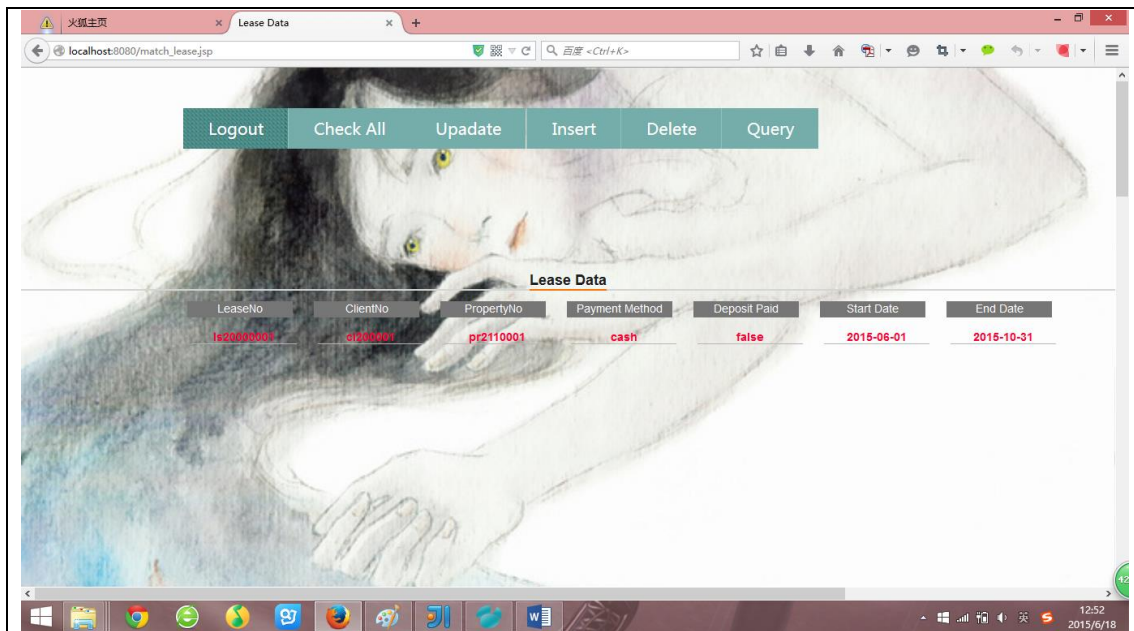
Display the details of a lease between a named client and a given property

pr2110001

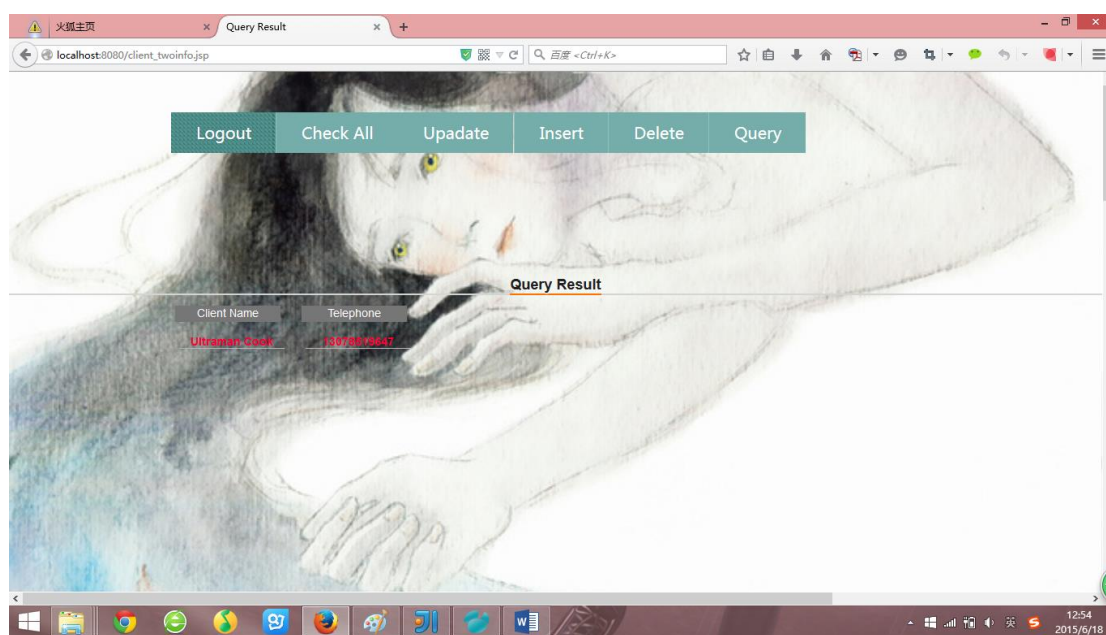
Bobby

Walker

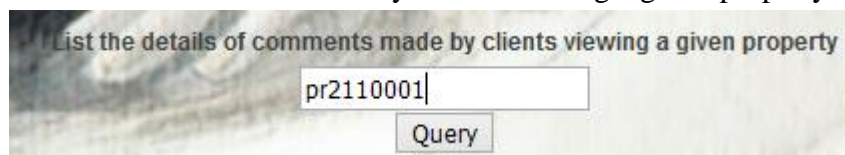
Query



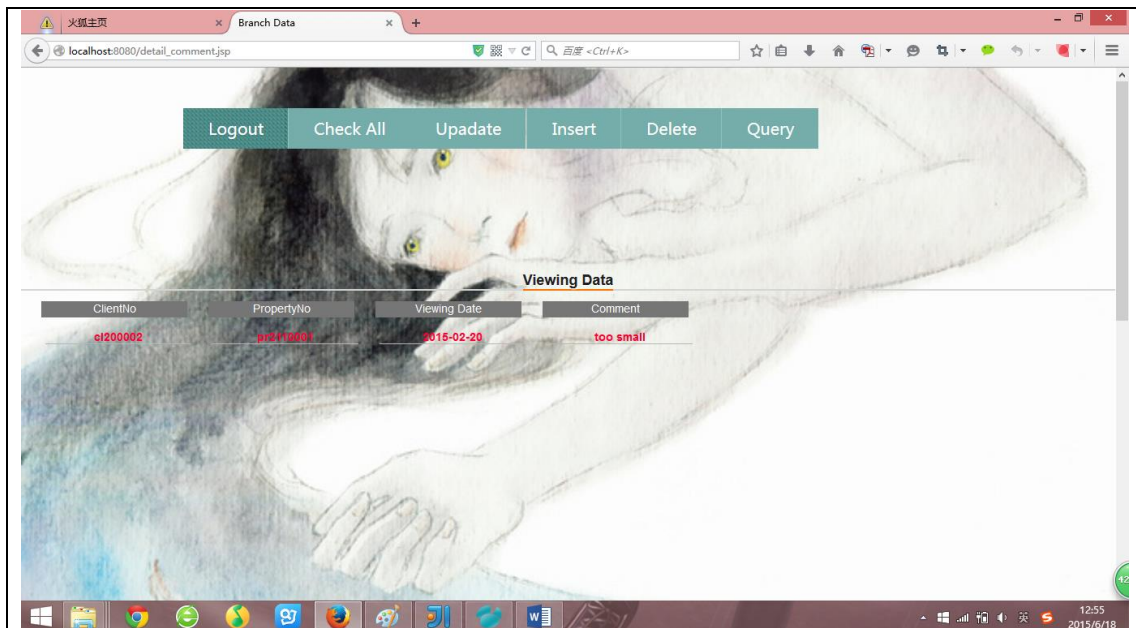
16. Display the names and phone numbers of clients who have viewed a given property but not supplied comments



17. List the details of comments made by clients viewing a given property

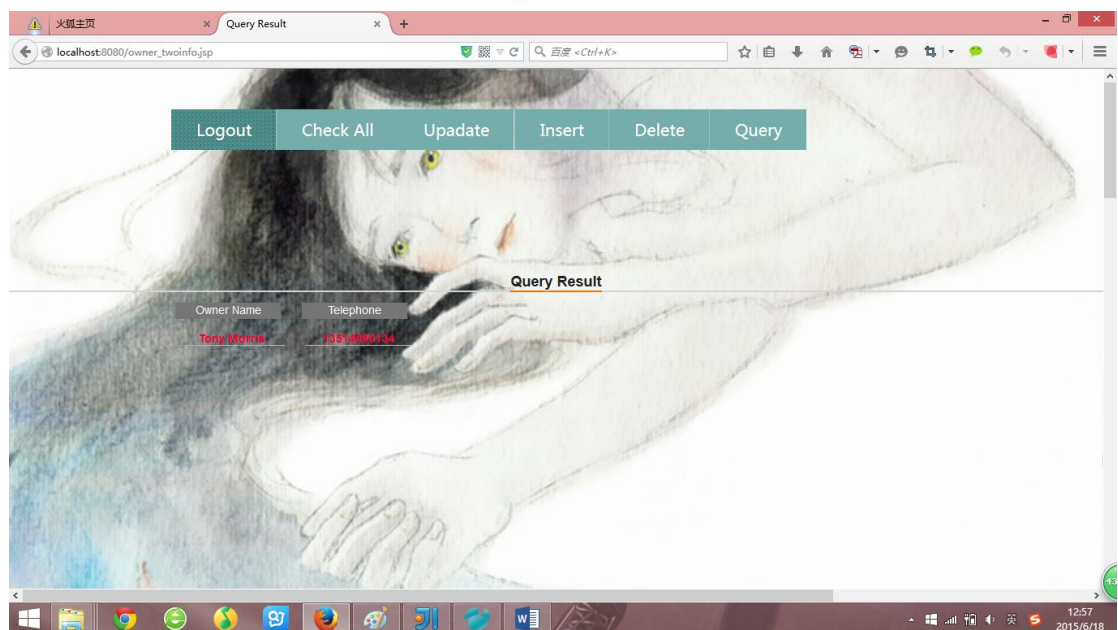






18. Identify the name and telephone number of an owner of a given property

Identify the name and telephone number of an owner of a given property



**【小结】：**

**From this experiment, we learned and practiced:**

- 1. Web Design and Web Development**
- 2. Struts2 and MVC**
- 3. Jsp, servlet, css and js**



4. JDBC

5. SQL manipulation

6. Solve problems on one's own or turn for others for help

7. IntelliJ IDEA usage

8. Tomcat configuration

指导教师评语及成绩

【评语】：

名：

成绩：

指导教师签

批阅日期：

附件：

## 实验报告说明

- 1. 实验项目名称：**要用最简练的语言反映实验的内容。要求与实验指导书中相一致。
- 2. 实验目的：**目的要明确，要抓住重点，符合实验任务书中的要求。
- 3. 实验环境：**实验用的软硬件环境（配置）。
- 4. 实验方案设计（思路、步骤和方法等）：**这是实验报告极其重要的内容。包括概要设计、详细设计和核心算法说明及分析，系统开发工具等。应同时提交程序或设计电子版。

对于**设计型和综合型实验**，在上述内容基础上还应该画出流程图、设计思路和设计方法，再配以相应的文字说明。

对于**创新型实验**，还应注明其创新点、特色。
- 5. 结论（结果）：**即根据实验过程中所见到的现象和测得的数据，做出结论（可以将部分测试结果进行截屏）。
- 6. 小结：**对本次实验的心得体会，所遇到的问题及解决方法，其他思考和建议。

**7. 指导教师评语及成绩：**指导教师依据学生的实际报告内容，用简练语言给出本次实验报告的评价和价值。